RELATÓRIO ESINF - SPRINT 3

DIAGRAMA DE CLASSES E COMPLEXIDADE DE ALGORITMOS

GRUPO 21 – 2DC

1191507 – Bárbara Pinto

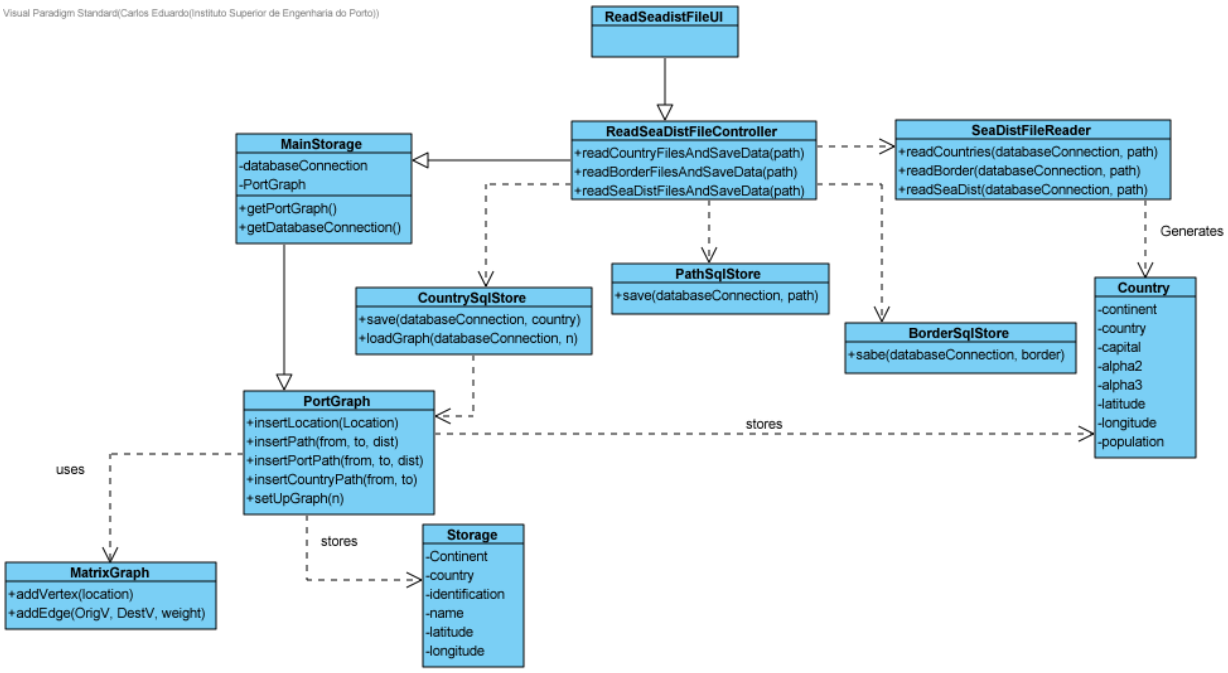
1200991 - Carlos Dias

1201029 - Cristóvão Sampaio

1201045 - Miguel Silva

1201154 – Martim Maciel

US301 – 100% - Carlos Dias 1200991



Esta User Story 301 pede para importar a partir de vários ficheiros os países, bordas dos países e as ligações entre portos diferentes e introduzir estas informações num grafo.

Para isso usaremos os diferentes *readers* criados para ler os vários ficheiros cada um deles tendo uma complexidade de $\boldsymbol{O(n)}$, esta complexidade é irrelevante pois a partir de agora usaremos grafos que usam a notação V- vértice e E – aresta. O gráfico depois criados usara uma classe abstrata de *Location* para diferenciar entre portos e capitais.

Como os valores lidos são guardados na base de dados podemos então usá-la para tornar a criação do ficheiro mais fácil. Começamos inicialmente por importar os países usando a classe *CountrySQLlStore* mas, sempre que um país é importado todos os seus portos são introduzidos logo de seguida, assim podes usar um algoritmo que calcula a distância da capital de dado pais aos seus restantes portos e assim criar uma ligação direta entre estes dois. A inserção de novos vetores no grafo passa pelos passos seguintes:

- Averigua se o vértice já se encontra no grafo usando método *indexof* com complexidade $\boldsymbol{O(V)}$, como é corrido V vezes a complexidade final é de $\boldsymbol{O(V^2)}$**;**

- Adiciona o vértice com complexidade $O(1)$, como acontece V vezes tem complexidade $\boldsymbol{O(V)}$;

- No pior dos casos verifica que a matriz é muito pequena para acomodar todos os vértices e por isso faz um *resize* que tem complexidade de $\boldsymbol{O}(\boldsymbol{E^2})$, como é corrida V vezes acaba por ter complexidade final de $\boldsymbol{O}(\boldsymbol{V} * \boldsymbol{E^2})$;

- Calcula a distância mínima da capital ao porto que, no pior caso, quando todos os portos pertencem a um único país tem complexidade de $\boldsymbol{O}(\boldsymbol{V})$**;**

- Insere um caminho de um porto para dado país em que ambos os vértices adicionados são validados tendo a complexidade de $\boldsymbol{O}(\boldsymbol{V})$, depois vai-se buscar o índex de cada vértice que têm complexidade de $\boldsymbol{O}(\boldsymbol{V})$ e finalmente adiciona a nova aresta ao sítio designado na matriz com complexidade $\boldsymbol{O}(\mathbf{1})$, como isto no pior caso pode acontecer $\frac{V}{2}$ vezes, se todos os países se ligarem excecionalmente a 1 porto a complexidade de pior caso seria de $\boldsymbol{O}(\boldsymbol{V^2})$;

Após serem introduzidos todos os vértices teremos de adicionar todas as arestas entre estes, para isso dividimos as ligações pelas bordas e pelos caminhos entre portos. Para ligar as Bordas recebemos a informação de cada ligação da base de dados e através dos nomes guardados nestas ligações vamos buscar os países cujos nome são iguais e de seguida, se os países forem encontrados dentro do grafo são adicionados os caminhos num processo de complexidade $\boldsymbol{O}(\boldsymbol{V})$ que é percorrido V vezes fazendo a complexidade final de $\boldsymbol{O}(\boldsymbol{V^2})$. A adição de caminhos entre portos é semelhante tem complexidade $\boldsymbol{O}(\boldsymbol{V^2})$;

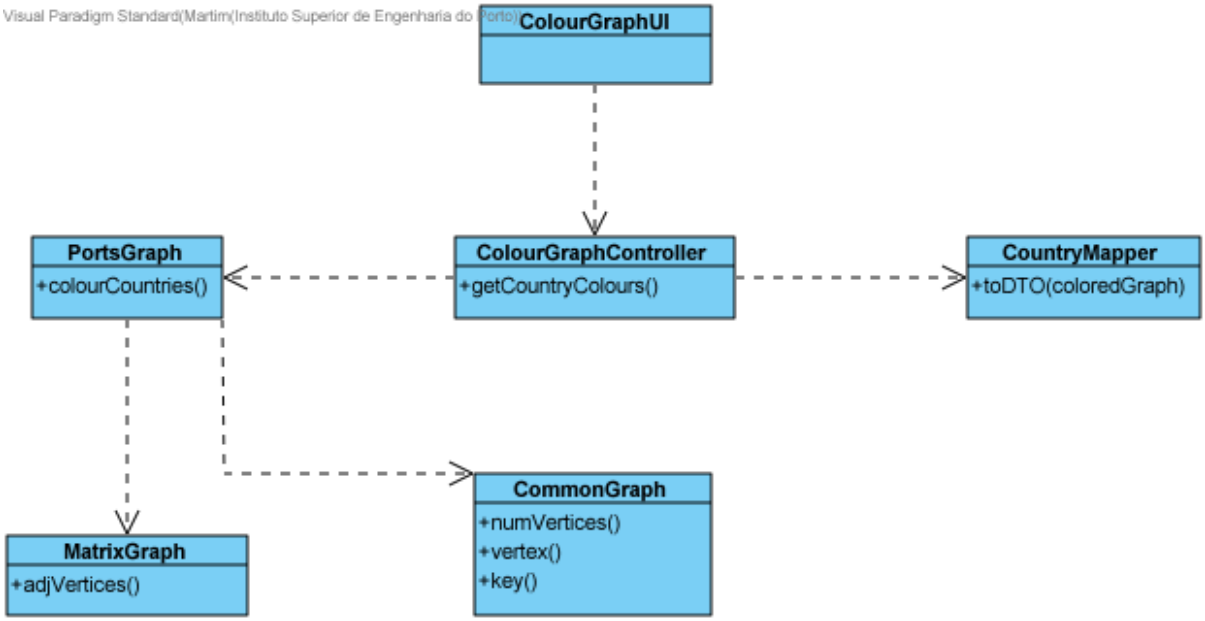
Finalmente é usado o método *setUpGraph* que é responsável por ligar todos os países a si mesmos, a todos os portos do mesmo país e aos n portos mais próximos de outros países.

Inicialmente este método cria uma lista de todos os portos existentes na matriz ignorando as capitais, iterando por todos estes e fazendo ligações de todos os vértices a si mesmos, esta criação de portos como é corrida $\boldsymbol{V}$ vezes e como adicionar arestas é uma operação de complexidade $\boldsymbol{O}(\boldsymbol{V})$ é de complexidade $\boldsymbol{O}(\boldsymbol{V^2})$**;**

De seguida este método cria um *loop* com todas os portos que foram guardados na lista previamente referida, dentro deste Loop ocorre outro *loop* que vai averiguar quais dos portos nessa lista são do mesmo país ou os que são de países diferentes, se forem do mesmo país é adicionada uma aresta se forem de outros países então serão adicionados a uma nova lista que guardara esta localização e a sua distância a localização inicial, finalmente esta segunda lista criada é ordenada pela distancia dentre as dua localizações e finalmente ás n localizações mais próximas são adicionadas arestas, para esta operação toda a complexidade é de $\boldsymbol{O}(\boldsymbol{V^3})$.

Esta User Story tem então na situação de maior complexidade O(V^3) e O(V*E^2) porem como na situação das arestas é impossível haver uma grande dimensão das mesmas podemos averiguar que o tempo de pior caso desta User Story é $\boldsymbol{O}(\boldsymbol{V^3})$.

US302 – 100% - Martim Maciel 1201154



A User Story 302 pede para a partir do gráfico criado na US anterior, que todos os países sejam coloridos de cores diferentes as dos que os bordam.

Para isso usamos um método chamado *GetCountryColours* que retorna uma *map* com todos os países e a sua respetiva cor. Inicialmente é criado um *array* que irá ser usado para guardar os valores de cor de cada país, e posteriormente é criado outro *array* de *boolean* que será usado na coloração do mapa, ambos os *arrays* são cheios por isso tem complexidade $\boldsymbol{O(V)}$.

É então feito um *loop* por todos os vértices do grafo em que primeiro e visto se o vértice a ser visto é uma capital, após isto é feito mais um *loop* com todos os vértices adjacentes a este vértice, neste *loop* o método vai buscar o índex de cada um destes vértices adjacentes e se eles estiverem coloridos a sua cor é dada como invalida no *array* de boolean anteriormente declarado, tanto a obtenção dos vértices adjacente como o *loop* por estes vértices são de complexidade $\boldsymbol{O(V)}$.

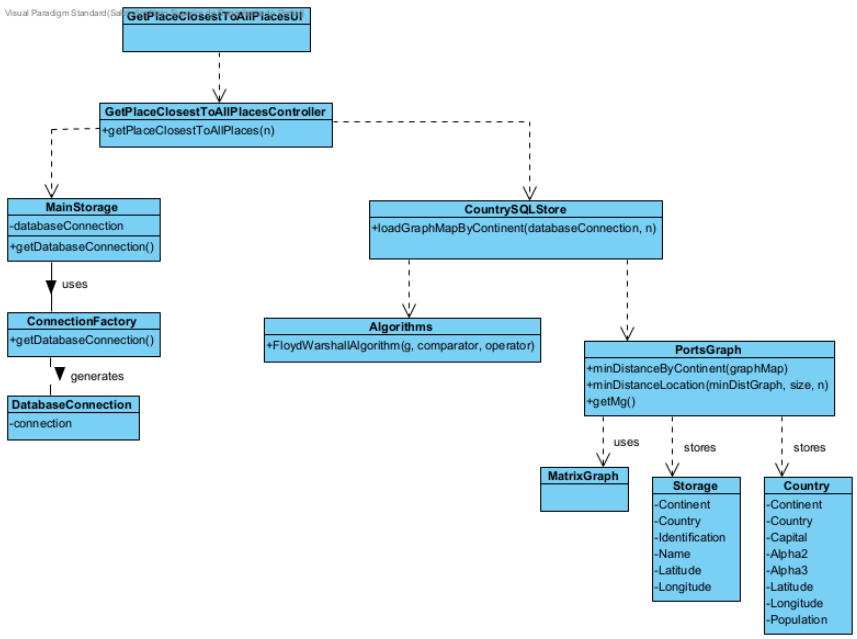
Finalmente faz-se um *loop* pelo *array* de *boolean* até encontrar o primeiro índex cuja cor n seja usada, essa cor acaba por ser a cor usada para pintar o dado vértice e finalmente o enchesse o *array* de *boolean* de novo tendo esta parte uma complexidade de $\boldsymbol{O(V)}$**.**

Como estes processos são corridos V vezes a complexidade final desta User Story é de $\boldsymbol{O(V^2)}$.

US303

50% - Cristóvão Sampaio 1201029

50% - Miguel Silva 1201045



A User Story 303 pede-nos para devolver várias listas com os *n* locais mais próximos de todos os outros locais pertencentes ao mesmo continente, agrupadas pelos continentes.

Para isso vamos primeiramente ter de utilizar aquilo que fizemos na US301 para criar vários grafos, grafos esses que contêm a informação dos locais de cada continente, ou seja podemos concluir que estamos a fazer a mesma coisa que fizemos na US301, sendo que o pior caso vai ser fazer essa operação 6 vezes no caso de todos os continentes estarem na base de dados (apenas são criados grafos com os continentes existentes na base de dados). Ou seja, podemos concluir que vamos ter uma complexidade de $\boldsymbol{O}(\boldsymbol{V^3})$ nesta operação.

De seguida vamos passar por cada um dos grafos criados anteriormente (que como concluímos que vamos repetir um máximo de 6 vezes não vamos contar para a complexidade) e vamos determinar quais os n locais que se encontram mais próximos de todos os outros locais. Para fazer isso vamos primeiro usar o algoritmo de *Floyd Warshall* que nos permite criar uma matriz com os caminhos mais curtos entre todos os vértices do grafo, isto tem uma complexidade de $\boldsymbol{O}(\boldsymbol{V^3})$.

Após ter a matriz dos caminhos mais curtos criada, vamos iterar por cada linha dessa matriz $\boldsymbol{O}(\boldsymbol{V})$ e calcular a média do peso dos caminhos, assim vamos ter uma média da distância a que cada vértice fica de todos os outros vértices (usando os caminhos mais curtos) e vamos colocando os resultados numa *list* $\boldsymbol{O}(\boldsymbol{1})$. Tendo isto em conta só nos falta organizar esta lista que criamos de forma crescente, sendo que para tal utilizamos o método *sort()* que tem uma complexidade de $\boldsymbol{O}(\boldsymbol{V} * \boldsymbol{log}(\boldsymbol{V}))$.

Por fim apenas temos de ir à lista criada anteriormente para ir buscar os *n* locais que estão no topo da lista $\boldsymbol{O}(\boldsymbol{n})$. Como este valor vai bem mais pequeno do que o número de vértices totais, podemos simplesmente ignorar esta complexidade.

Podemos assim concluir que a US303 vai ter uma complexidade de $\boldsymbol{O}(\boldsymbol{V^3})$.